Automatic Construction of Interactive Machine Improvisation Scenarios from Audio Recordings

Jaime Arias, Myriam Desainte-Catherine, Shlomo Dubnov

Automatic Construction of Interactive Machine Improvisation Scenarios from Audio Recordings

Jaime Arias and **Myriam Desainte-Catherine**
Univ. Bordeaux, LaBRI, UMR 5800, F-33400 Talence, France
CNRS, LaBRI, UMR 5800, F-33400 Talence, France
Inria, F-33400 Talence, France
{jaime.arias, myriam.desainte-catherine}@labri.fr

Shlomo Dubnov
CREL, Music Department
University of California, San Diego
9500 Gilman Dr., La Jolla, CA 92093
sdubnov@ucsd.edu

Abstract

We describe a system that allows improvisers and composers to construct an interactive musical environment directly from a musical recording. Currently, interactive music pieces require separate phases of constructing generative models and structuring them into a larger compositional plan. In the proposed system we combine machine improvisation tools based on *Variable Markov Oracle* (VMO) with an interactive score (i-score) to control the improvisation according to larger structures found in that recording. This allows construction of improvisation scenarios in ways that are organic with the musical materials used for generating the music. The method uses new results of audio segmentation based on VMO and translates it into a *Petri Net* (PN) model with transition rules left open to be defined by a musician. The PN structure is finally translated into a timed representation for a live i-score control.

Introduction

Musicians working in the area of machine improvisation use machine analysis tools to generate a computer accompaniment from musical examples. This allows the computer to generate music in ways that preserve the style of the musical examples given to it without a need for programming. During performance, these generative mechanisms are either controlled in real-time by another operator or are programmed to interact according to readily prepared scenarios. In this work we use machine learning tools to add a second layer of analysis to machine improvisation that automatically constructs an initial sketch of a possible structured improvisation that follows short-term (surface) stylistic features, and also observes initial long term structures (form) of the materials used for improvisation. The automatic construction of such two-level structural model from a recording gives a significant head start to musicians working creatively with musical materials using computational tools. In the following we briefly describe the technologies involved in this research and provide an example of such a use on a pre-recorded piece.

Preliminaries

In the following we briefly introduce the formalisms and tools used in our approach.

Variable Markov Oracle

Variable Markov Oracle (VMO) (Wang and Dubnov 2015) is the newest development in a family of machine improvisation methods based on automatic analysis of the musical structure of a recording in terms of its suffix structure and subsequent recombination of segments of that recording to creative variations. Allowing transitions within a prerecorded musical material that preserve partial suffix overlap allows creating novel variations with smooth and natural continuations. VMO extends previous work on *Audio Oracle* (AO) (Dubnov, Assayag, and Cont 2011) in two principal ways – it allows construction of an optimal model by adaptive symbolization of audio features, which in turn allows segmentation or partitioning of the model into regions of similar musical materials, thus providing a structural analysis of the whole piece. VMO represents the audio recording in terms of a graph structure that can be used for query-guided audio content generation and multimedia query-matching. VMO is capable of finding common suffixes between samples along the multivariate time series and enables us to devise various search algorithms on that data structure. Moreover, graph partitioning methods can be used to find densely connected areas in the graph that correspond to areas in the recording that have many similar phrases of various lengths. A spectral clustering method of such partitioning is used in this paper (Wang and Mysore 2016).

Petri Net

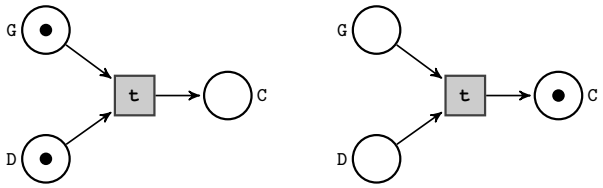
Petri Net (PN) (Murata 1989) is a graphical language for description and analysis of concurrent and distributed systems. It provides useful visual tools to easily model, interpret and analyze complex systems with parallelism, concurrency, synchronization and resource sharing. Additionally, it can be used to represent not only control flow but also data flow.

PN provides a *compact representation* of systems with a very large state space, allowing to represent systems with an infinite number of states using a finite structure. Intuitively, a PN is a directed, weighted, bipartite graph consisting of two types of nodes: *places* (represented by circles) and *transitions* (represented by rectangles). Each place may hold ei-

ther none or a positive number of tokens (represented by small solid dots). Directed arcs (represented by arrows) connect places to transitions and transitions to places. Arcs are labeled with a positive weight k which can be interpreted as a set of k parallel arcs.

A state of a PN (*i.e.,* marking) is represented by the number of tokens assigned to each place. In order to simulate the dynamic behavior of a system, a state in a PN is changed according to a *firing rule*. For instance, in a basic PN a transition `t` is said to be *enabled* if each input place of `t` contains at least one token. An enabled transition fires depending on whether or not a specific event takes place. The firing of an enabled transition `t` removes a token from each input place of `t`, and adds a token to each output place of `t`.

Let us illustrate the notion introduced above with the PN shown in Figure 1. Assume that the transition `t` has two input notes G and D in the input, and an output that plays C. Then, this PN creates a melodic sequence that could be either G-D-C, or D-G-C. For that, the transition waits until both notes G and D occurred (*i.e.,* the transition is enabled as shown in Figure 1a) and then the resolution C is played (*i.e.,* the transition is fired and not longer enabled as shown in Figure 1b).



(a) The marking before firing the enabled transition `t`.

(b) The marking after firing transition `t`, where `t` is disabled.

Figure 1: Illustration of a firing rule in a PN.

In the literature exists several extensions of the PN model. For instance, in this work we use *Time Petri Net*s (TPNs) (Merlin and Farber 1975) which deal with *time*. Intuitively, a TPN is defined by a PN where each transition has two times specified. The first denotes the minimal time that must elapse from the time that all the input conditions of a transition are enabled until this transition can fire. The other time denotes the maximum time that the input conditions can be enabled and the transition does not fire. After this time, the transition must fire. In general, these two times give some measure of minimal and maximal execution times of the transitions.

We use the SNAKES[1] Python library (Pommereau 2008) for prototyping our approach that is presented in the next section. Roughly, SNAKES is a general PN library that allows to model and execute PNs where tokens are Python objects, and net inscriptions are Python expressions. Moreover, SNAKES provides a plugin system that allows to easily extend its built-in features. For instance, the reader can found the SNAKES plugin for modeling TPNs at `https://www.ibisc.univ-evry.fr/~fpommereau/blog/2016-04-25-time-petri-nets-with-snakes.html`.

[1] `https://github.com/fpom/snakes`

The inter-media sequencer i-score

I-score[2] is an open source inter-media sequencer that aims to provide an integral environment for writing and performance of interactive multimedia scores (de la Hogue, Celerier, and Baltazar 2016).

This tool supports branching behaviors allowing the composer to define conditions and loops in order to modify the execution path of the score during its performance. The composer also has the ability to define the intervals of time in which the performer or external devices may or may not interact with the system. This interaction is done by means of OSC[3] messages and only authorizes *agogic modifications* (Haury 1987): the possibility to change the times of beginning and end of the multimedia content.

I-score provides a graphical interface based on a time-line model that allows the user to write complex pieces in an intuitive way. Contrary to other interactive score systems like INSCORE (Fober, Orlarey, and Letz 2014), which is able to handle only musical objects, i-score can control any multimedia object that is accessible via OSC. For instance, in the approach presented in this paper, we use i-score to orchestrate the improvisation carried out by an AO given a segmentation of an audio recording.

Let us now briefly introduce the elements provided by i-score for the specification of interactive scores. For that, we shall use the interactive score presented in Figure 2 that specifies the score of an improvisation controller system. We refer the reader to (de la Hogue, Celerier, and Baltazar 2016) for further details on the semantics of i-score.

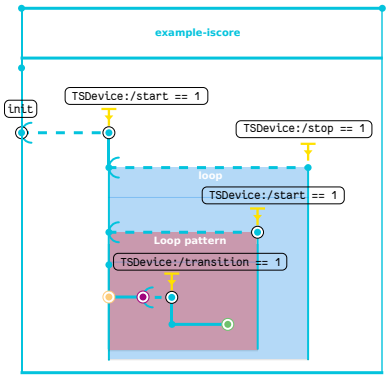


Figure 2: Example of an interactive score in i-score that controls an improvisation system.

Assume that the improvisation system reacts to three events from the user. The first event indicates the *starting* of the improvisation, the second event announces a *transition* to a new musical material, and the third event indicates the

[2] `http://www.i-score.org/`

[3] Open Sound Control (OSC) is a protocol for communication among multimedia devices (`http://opensoundcontrol.org/`).

ending of the improvisation. When the improvisation starts, the systems improvises on the musical material *yellow* immediately, and after 1 minute it changes to the musical material *red*. The improviser must improvise on material *green* when both the user sends the event for the transition and the improviser is currently improvising on material *red*. This transition takes effect 2 minutes after the user has sent the event. The above behavior is repeated until the user decides to stop the improvisation.

In order to specify this score in i-score, we assume that the user and the improviser interact with i-score by means of OSC messages. We recall that i-score uses a time-line model in which time passes from left to right. The first element that we shall introduce is the *event*. Events are points in time that may (◉) or may not (●) contain a set of OSC messages that are sent to the environment when the event is executed. For instance, in Figure 2 we define an event at 0 s, labeled as `init`, in order to send a message to the improviser for setting its initial parameters (*e.g.,* tempo).

Events are temporally organized by adding temporal *intervals* (—{ - -}) between them. Intuitively, the solid line of the interval represents the minimum amount of time that should elapse in order to execute the event. However, the user can define a possibly infinite range of times (denoted by a dashed line) in which an *interaction* (Ŧ) may or may not occur in order to execute the event. That interaction happens when a logical expression specified by the user is satisfied within the defined interval of time. For instance, in Figure 2 the performer must send to i-score an OSC message indicating the starting of the improvisation (*i.e.,* `TSDevice` : /`start` == 1) in order to execute the box following the initial event. It is important to note that this interaction can occur at any time since the minimum time is 0 and the range of interaction is not bounded (*i.e.,* infinite).

I-score also allows to define loops. For that, the content to be repeated (*i.e.,* the box labeled as `loop pattern`) is encapsulated into a temporal structure (*i.e.,* the box labeled as `loop`) whose duration is defined by a temporal interval. The content to be repeated also has a temporal interval defining when the content should be repeated. For instance, in our score changing the musical materials is an action which is repeated each time the user sends the starting event (*i.e.,* `TSDevice` : /`start` == 1) until the stopping event is sent (*i.e.,* `TSDevice` : /`stop` == 1).

Therefore, each time the starting event is triggered, an OSC message is sent to the improviser in order to improvise on the musical material yellow (*i.e.,* yellow event ◉) and then, using a temporal interval, we change the musical material to red after 1 minute (*i.e.,* red event ◉). Once the user sends the transition event (*i.e.,* `TSDevice` : /`transition` == 1), we wait for 2 minutes to change the musical material to green (*i.e.,* green event ◉). As we explained before, this pattern is repeated each time the start event is sent until the user stops the improvisation.

Approach

Now we shall introduce our system for building interactive improvisation scenarios from an audio recording. It provides three stages: *segmentation*, *offline improvisation*, and *real-time improvisation*. For better understanding, we describe each stage using as a running example the audio file *Luciano Berio: Sequenza I* (Priore 2007). The reader can find the files of the tool and examples at `http://himito.github.io/vmo_i-score_generator`.

Segmentation stage

As shown in Figure 3, we firstly use VMO to generate, from a pre-recorded audio, the AO for improvisation. In addition, as is pointed out in (Wang and Mysore 2016), we take advantage of the spectral clustering method provided by VMO to recognize repetitive/homogeneous structures from the audio file (*i.e.,* segmentation analysis). Doing this, we are able to improvise on regions whose musical content is related and perceive possible transitions on the material during improvisation.

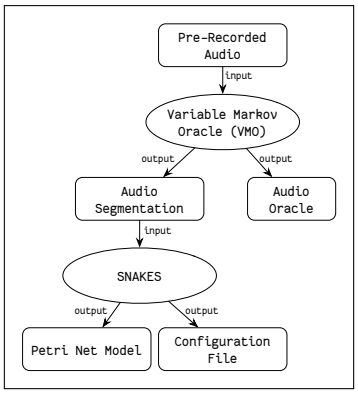


Figure 3: Overview of the segmentation stage.

As an example, we present in Figure 4 the segmentation analysis of our running example. For the sake of clarity, each section was labeled by color representing similarity among larger sections of this work. Note for example that the section *gray* has no similar musical content throughout the work. In that sense, we may label that section as the introduction material of the work. Also, we can notice that during improvisation a more organic transition to material of section *red* is only possible from material of section *green*.

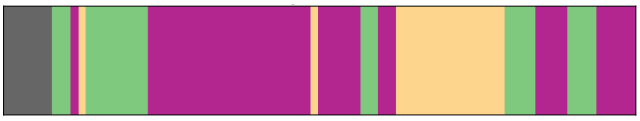


Figure 4: Segmentation analysis using VMO of the audio file *Luciano Berio: Sequenza I*.

As we remarked above, the segmentation analysis allows the identification of possible organic transitions between sections with similar musical content. However, the representation depicted in Figure 4 does not facilitate the reasoning about the logical and temporal organization of the material for a possible improvisation. To alleviate this problem, we define a *Time Petri Net* (TPN) representation of the

musical structure described by the segmentation analysis in order to provide a higher, more intuitive and formal representation of this structure. Moreover, the composer can now define temporal constraints and performer interactions to the improvisation by adding temporal and logical conditions to the PN transitions.

To translate the segmentation structure into a PN, we first convert each section label of the segmentation analysis (*e.g.,* a color in Figure 4) into a PN place. That is, a place denotes all frame intervals belonging to the sections corresponding to a label. Next, for each local change point from section s_1 to s_2 in the segmentation analysis, we add a transition from the place p_1 to the place p_2 that contains the section s_1 and s_2, respectively. Finally, we add a transition from (*resp.* to) the place containing the last (*resp.* first) section in the segmentation analysis to a new place denoting the end (*resp.* start) of the improvisation.

As an example, we illustrate in Figure 5 the PN representation of the segmentation analysis depicted in Figure 4. Observe that each section label (*i.e.,* colors gray, green, red and yellow) is represented by a PN place. Moreover, the place denoting the last section (*i.e.,* red place) has a transition to a place with no outgoing transitions (*i.e.,* end of improvisation), and the place denoting the first section (*i.e.,* gray place) has a transition from a place with no ingoing transitions (*i.e.,* start of improvisation).

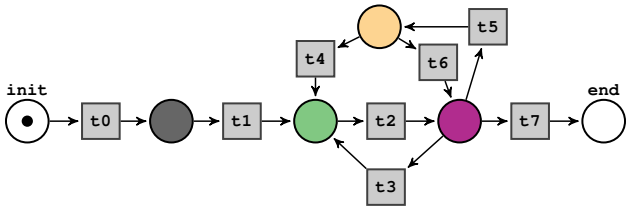


Figure 5: Petri net built from the segmentation analysis shown in Figure 4.

Now, by observing the structure of the Petri net it is more intuitive for the artist to know what are the possible more organic transitions between sections. Moreover, features such as loop sequences (*e.g.,* transitions `t2`, `t5` and `t4`) or sequences leading to the end (*e.g.,* all possible paths leading to the sequence red-end) are more intuitive to find out. We can experiment with different configurations and also reason about the generated PN structure using the SNAKES library. For instance, we may experiment with a PN in which it is possible to have a token in different places at the same time, meaning that the oracle improvises on different sections simultaneously.

As we claimed before, the artist can now control the improvisation by defining in a *configuration file* temporal constraints and interactions with the environment (*e.g.,* the performer) on the PN transitions. For instance, the artist may impose the constraint that the oracle will improvise on the material belonging to the introduction (*i.e.,* gray place) for 3 minutes as maximum and while a specific pedal is not pressed.

Offline improvisation

Once the artist has defined the parameters of the PN model, it is possible to execute it offline using SNAKES in order to generate a timed sequence of frame intervals. Since the PN is a finite structure that allows to specify infinite and non-determinist behavior, then we can create a potentially infinite number of new sequences with the same PN that satisfy the constraints imposed by the composer.

Recall that the firing of some PN transitions may be conditioned by the interaction with the performer. Therefore, the artist can define in the configuration file a possible behavior of the performer (*i.e.,* actions) in order to avoid getting stuck in one place. Additionally, the artist must define the duration of the resulting sequence because the execution of the PN could not finish due to loops and the conditions on transitions.

After generating the sequence, we use the functions provided by VMO to improvise on the AO created during the segmentation stage. The improviser must improvise on the frame interval (*i.e.,* musical material) imposed by the sequence. Moreover, the sequence also defines the time and the duration that the improviser must keep improvising on the same material. As shown in Figure 6, the offline improviser takes from the configuration file the parameters of the oracle (*i.e., continuity* and *lrs*) defined by the artist.

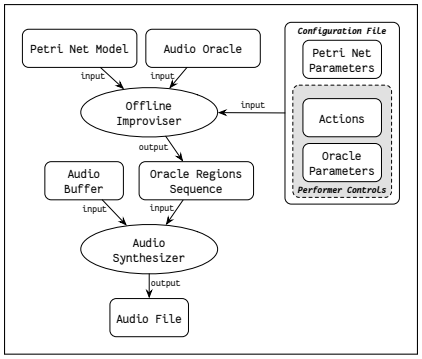


Figure 6: Overview of the offline improvisation stage.

Finally, we synthesize the audio from the sequence of oracle's regions generated by the improviser. For that, we use VMO to link the frames of the audio buffer of the AO with the frames in the generated sequence, and save the output in a file.

Real-time improvisation

A drawback of the PN representation is that it does not provide a real-time platform for improvisation and interaction. In addition, the modification of the PN structure could be cumbersome for artists. To overcome these problems, we thus propose a mechanism to encode the PN representing the segmentation analysis as an i-score's interactive scenario.

In Figure 7 we present the architecture of the real-time improvisation stage. As we pointed out before, i-score allows to orchestrate any device that uses the OSC protocol. In this regard, we equipped PyOracle – an AO system pro-

posed in (Surges and Dubnov 2013) – with OSC communication for the sake of improvising in real-time following the orders sent by the i-score scenario.

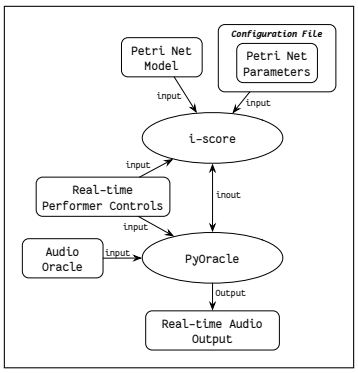


Figure 7: Overview of the real-time improvisation stage.

Recall, that this scenario represents the structure of the improvisation desired by the artist (*i.e.,* the PN model) and it can be modified by the performer in order to add, modify or remove constraints and interactions. Observe that in our approach, the performer also may or may not interact with the scenario and modify the parameters of the oracle. Then, in the absence of any interaction, the execution of the scenario would be the defined by the composer as default.

To translate a PN structure pn into an i-score score, we follow Algorithm 1. Roughly, we firstly add an initial i-score constraint denoting the PN transition of the initial PN place (*i.e.,* ti). This constraint starts at the beginning of the score and it has an i-score interaction whose parameters are defined by the temporal interval and the interaction condition of the PN transition.

Algorithm 1 Translate a Petri net into an i-score score

procedure PN_ISCORE(pn)
  $ti \leftarrow$ `pos_t`(`initial_place`(pn));
  $te \leftarrow$ `pre_t`(`final_place`(pn));
  $ei \leftarrow$ `add_constraint`(`start_score`, `min_dur`(ti), `max_dur`(ti), `cond`(ti));
  for all $p \in$ `c_places`(pn) **do**
    $pre \leftarrow$ `pre_t`(p);
    $pos \leftarrow$ `pos_t`(p);
    $el \leftarrow$ `add_loop_flexible`(ei, `or`(pos), te);
    $ec \leftarrow$ `add_constraint`(el, 0, ∞, `or`(pre));
    for all $t \in pos$ **do**
      `add_constraint`(ec, `min_dur`(t), `max_dur`(t), `cond`(t));
    end for
  end for
end procedure

Then, we add an i-score loop after the above constraint (*i.e.,* ei) for each colored place p in the PN. Each loop has no minimum and maximum durations. Moreover, the stop condition of each loop is the triggering of the final transition (*i.e.,* te) while the looping condition is the triggering of one of the outgoing transitions (*i.e.,* disjunction of pos) of the place p.

Inside each loop, we add a constraint starting at the beginning of the loop (*i.e.,* el) with an i-score interaction with no minimum and maximum durations and whose condition is the triggering of one of the ingoing transitions (*i.e.,* disjunction of pre) of the PN place. Finally, we add for each outgoing transition t of the PN a constraint starting from the above constraint (*i.e.,* ec). Each constraint has an interaction whose parameters are defined by the temporal interval and the interaction condition of the transition t.

We clarify the above with Figure 8 that illustrates the i-score translation of the PN in Figure 5. Observe that the colored events represent the colored places in the PN. Moreover, intervals representing PN transitions are labeled with the same name as in the PN.

Conclusions and further work

In this work we presented a new method for improvised interaction with recorded musical materials that combines style learning properties of PyOracle and interactive scenario manipulation using a combination of i-score with segmentation done by VMO. Moreover, we also presented an offline version of the method based on PN, and a new PN loader to i-score, so that the same musical piece could be either composed or tested offline, or performed live with i-score.

The overall approach to used of machine improvisation in this project falls under the broad scope of structured improvisation or composition with aleatoric or indeterminate elements. This approach to improvisation practice differs from the traditional use of tools like OMax and PyOracle, since the analysis of the musical input that is used as the basis for improvisation is done on pre- recorded materials. This is contrasted with another common use of machine improvisation that uses the online nature of the machine learning algorithms that allows construction of the improvisation model on the fly from the music performed by the human musicians live. Since the VMO segmentation is done on a complete recording, at this point our system can not be used to construct or add to the improvisation scenario during performance. Such development should be possible in principle by using online change detection and clustering algorithms and allowing incremental updates to i-score in run time.

Besides the technical aspects of the integration of the different software tools for music improvisation with interactive scenario management, the current system also offers a new framework and paradigm for music composition. In our model we combine two conceptual approaches - one of stochastic time series modeling that captures surface properties of musical materials, with a process management of dynamic systems by the PN model. The role of the composition or improvisation planning becomes one of creating the interaction points and synchronization conditions between the live musician or the musical score, and the machine generative process of the PyOracle.

Conceptually one can view the live musician both as part of the environment and as another dynamic process, where

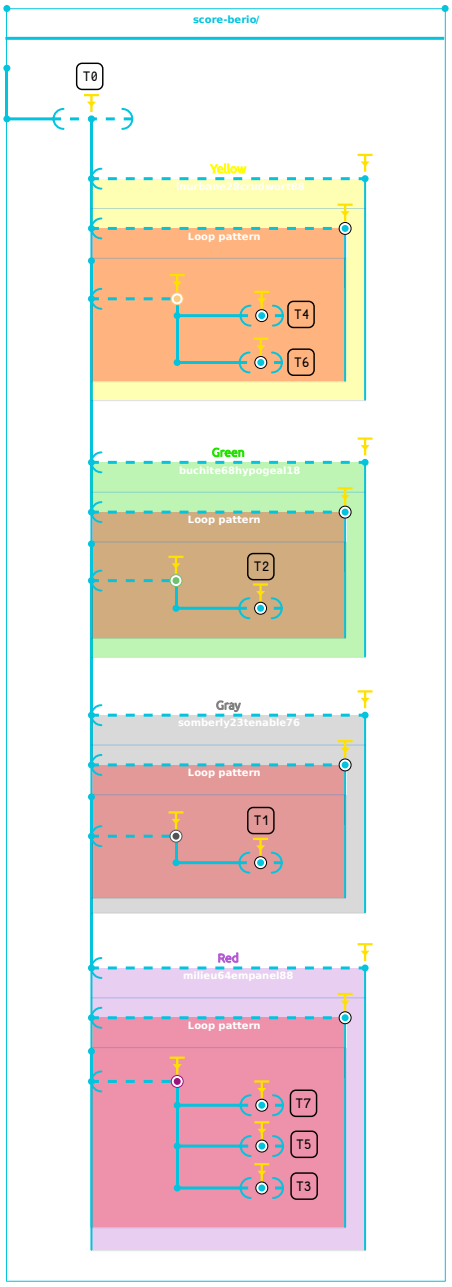


Figure 8: Representation in i-score of the Petri net depicted in Figure 5.

the compositional decisions are formally represented as conditions and actions that provide concurrency control between human and the robotic improvisation processes. The level of synchronization and the division of surface and background structure in music is formally divided into improvisation based player that operates on different musical materials as captured by states of the PN or loops in i-score, and transition conditions the align it with the human musical line on the level of musical form.

The extend of influence the musician has on generation of PyOracle within a given state are limited to general machine improvisation parameters such as probability of continuation or rate of recombination and smoothness of the recombinations as function of the common memory length between recombination segments. There are additional surface level synchronizations possible, such as limiting the recombination to common beat or guiding the improvisation to a certain tonal center, or even a specific chord in a chord progression in case it is know in advance or estimated live from a human input.

PyOracle currently has a query function that allows creation of "hot spots" or higher weight probabilities of recombination jumps to regions that correspond to some feature of the external live music input. Beat and tempo synchronization and activation of query functions to guide the machine towards generating materials that have higher cohesiveness with an external source are currently under development.

Another method for use of our system is in the case of a pre-defined scenario. A composer may enter a sequence of desired states, such as a chord progression, with approximate flexible timings. This scenario can be added as an additional element in the environment, with controls for precise timing of the transitions left to be determined by the users as performance conditions and actions. In addition to synchronizing the timing, the system may also listen to other user performance data within each state and providing a set of secondary or supplemental controls. Such control will not only affect the PyOracle generation by weighting trajectories or recombinations within a state, but may also guide the choices of jumps on the oracle graph from one PN state to another when a PN transition occurs.

We plan to introduced path searching algorithm to label possible oracle trajectories across states in order to allow controlling aspects of articulation during state transitions, or in other words we plan to find way to determine what happens on the musical surface (the detailed generation of materials by PyOracle) when a state transition occurs by having the system "listen" to secondary parameter, such as expressive aspects of the performance. This method is motivated and extends the ideas of the ImproteK (Nika and Chemillier 2014) system that allow improvisation with pre-determined scenarios.

Last and most challenging aspect of future research is creating dynamic scenarios through establishing discourse rules. In such usage the transitions will be determined autonomously by a set of rules that will choose the most appropriate transition or choice of materials by the machine in response to music played by the human improviser. For example, the system might decide to move to a state that has most similar or most contrasting music materials relative to music performed by the live musician. In such case the timing of the state is less significant and the PN will navigate autonomously to the best state or move through a sequence of states as a response to the nature of the external music it "hears" at every moment. The development of such machine listening elements that will translate external musical input to conditions and actions needed for PN transitions, as well as creating discourse rules to map conditions to actions is left for future work.

Acknowledgments

We thank the anonymous reviewers for their detailed comments that helped us to improve this paper. Also, we would like to thank Jean-Michaël Celerier for his help with i-score, Cheng-i Wang for his help with the VMO segmentation code and Franck Pommereau for his help with SNAKES. This work has been supported by the IdEx visiting scholar program of the University of Bordeaux, the PoSET project[4], SCRIME[5], and CREL[6].

References

[de la Hogue, Celerier, and Baltazar 2016] de la Hogue, T.; Celerier, J.-M.; and Baltazar, P. 2016. Presentation d'un Formalisme Graphique pour l'Ecriture de Scenarios Interactifs. In *JIM 2016, Albi, France*, 37–41.

[Dubnov, Assayag, and Cont 2011] Dubnov, S.; Assayag, G.; and Cont, A. 2011. Audio oracle analysis of musical information rate. In *ICSC*, 567–571. IEEE.

[Fober, Orlarey, and Letz 2014] Fober, D.; Orlarey, Y.; and Letz, S. 2014. Augmented Interactive Scores for Music Creation. In *Korean Electro-Acoustic Music Society's*.

[Haury 1987] Haury, J. 1987. La Grammaire de l'exécution musicale au clavier et le mouvement des touches. *Analyse musicale* 7:10–26.

[Merlin and Farber 1975] Merlin, P. M., and Farber, D. J. 1975. Note on Recoverability of Modular Systems.

[Murata 1989] Murata, T. 1989. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE* 77(4).

[Nika and Chemillier 2014] Nika, J., and Chemillier, M. 2014. Improvisation musicale homme-machine guidée par un scénario temporel. *Technique et Science Informatiques* 33(7-8):651–684.

[Pommereau 2008] Pommereau, F. 2008. Quickly prototyping petri nets tools with SNAKES. In *SimuTools*, 17. ICST/ACM.

[Priore 2007] Priore, I. 2007. Vestiges of twelve-tone practice as compositional process in berios sequenza I for solo flute. In Halfyard, J. K., ed., *Berio's Sequenzas: Essays on Performance, Composition and Analysis*. Ashgate. 191–208.

[Surges and Dubnov 2013] Surges, G., and Dubnov, S. 2013. Feature Selection and Composition Using PyOracle. In *AAAIDE Workshop*, 114–121.

[Wang and Dubnov 2015] Wang, C.-i., and Dubnov, S. 2015. The Variable Markov Oracle: Algorithms for Human Gesture Applications. *IEEE MultiMedia* 22(4):52–67.

[Wang and Mysore 2016] Wang, C., and Mysore, G. J. 2016. Structural segmentation with the variable markov oracle and boundary adjustment. In *ICASSP*, 291–295. IEEE.

[4] http://www.inria.fr/equipes/poset

[5] http://scrime.labri.fr/

[6] http://crel.calit2.net/